Exercise Set

Principles of Computing

Week 2

1 *Exercises*

shell

1. Suppose you just open up a fresh terminal instance.
   1.1. Where are you?[1]
   1.2. How do you check to see where you are?
   1.3. How can you see what files are in the current directory?
2. Now suppose we run the command `lss`. What should happen?
3. Now suppose we run the command `cd/`. What should happen?
4. What output should we expect to see if we executed the following?

```
cd /
pwd
cd ~
mkdir testing_dir
cd testing_dir
touch this_is just_a test_case
ls
```

[1] Meaning: what is the full path to where the shell running right now?

Python

1. What would be the result of running the following block of code?

```
def my_first_function():
  print("Hello, world!")
```

2. Suppose we define the following function.[2]

```
def current_score(x, y):
  irish_points = x + 1000000
  niu_points = y + 0
  print("ND points:", irish_points)
  print("NIU points:", niu_points)
```

   2.1. How do we *call* this function on the inputs `14` and `6`?[3] What should we expect to see as a result of running this function?
   2.2. What if we ran it on the inputs `"0"` and `6`?
   2.3. What about the inputs `"7"` and `"7"`?

[2] *Note:* the `print` function can take more than one input; in such cases, we separate the multiple inputs to `print` with commas, as on lines 4 and 5 of the snippet.

[3] *Calling* a function on some inputs is synonymous with *running it with those inputs*.

3. Suppose we define the string `mystring = "Irish by a million!"`
   3.1. How do we obtain the *first* character of this string?
   3.2. How do we obtain the *last* character of this string?
   3.3. How do we obtain the character in the *middle* of this string?

2 *Solutions*

shell

1. 1.1. A fresh terminal should start you off in your *home directory*, which has the shortcut name `~` for convenience. If your username is `username`, then the full path to home is `/Users/username` on `macOS` and `/home/username` on `Linux`.

   1.2. `pwd` displays the path to the working directory.[1]

   1.3. `ls` displays the files and folders in your working directory.

2. We should see one of the two following error messages, depending on if we are running `zsh` or `bash` as our shell.[2]

```
zsh: command not found: lss
```

```
bash: command not found: lss
```

3. We should see one of the two following error messages, depending on if we are running `zsh` or `bash` as our shell.[3]

```
zsh: no such file or directory: cd/
```

```
bash: cd/: No such file or directory
```

   Since the first thing the shell expects to see on a line is the *name of a command*, with commands and input arguments *separated by spaces*, the shell looks for a command named `cd/`. There is no such command.[4] When the shell can't find a command, it would normally throw a `command not found` error—as in the previous example; however, in this instance, the shell notices that there is a `/` character in the command you tried to run. The shell now tries to interpret `cd/` as a *path*, starting in the current directory, to a file or command that it should try to run. Of course, since there is no such file or directory,[5] the shell finally gives up and throws the error message we saw above.

4. In that example, only the `pwd` and `ls` commands display any output to the terminal. You should see the following output in order.

```
/
just_a    test_case this_is
```

Python

1. This code *defines* a function with the name `my_first_function`. *That's all this block of code does.* If we were to *run* the function, then it would

[1] Your *working directory* is what we call the current location of your shell.

[2] `zsh` is the default shell on newer versions of `macOS`. `bash` is the default shell on `Linux` and older versions of `macOS`.

[3] `zsh` is the default shell on newer versions of `macOS`. `bash` is the default shell on `Linux` and older versions of `macOS`.

[4] unless you've done something *strange* to your computer

[5] unless you've done something *horrible* to your computer

print the message `"Hello, world!"` to the terminal, but *we didn't run the function.* All we did was *define* it. Therefore, running this block of code *will not display anything to the terminal.*

2. 2.1. We call a function by invoking its *name* followed by *parentheses,* with the *inputs* to the function given inside the parentheses *separated by commas.* To call this function on `14` and `6`, we write the following line of `Python` code.

```
current_score(14, 6)
```

Running this, we should see the following output in the terminal.

```
ND points: 1000014
NIU points: 6
```

2.2. After a stack trace,[1] we should see a `TypeError`.

[1] We will talk about this later.

```
TypeError: can only concatenate str (not "int") to str
```

2.3. After a stack trace,[2] we should still see a `TypeError`, as before.

[2] We will talk about this later.

```
TypeError: can only concatenate str (not "int") to str
```

3. 3.1. Remember that indexing in `Python` begins at 0.

```
mystring[0]
```

3.2. The simplest way involves *negative indexing*.

```
mystring[-1]
```

However, we could explicitly get the last index instead.

```
last_index = len(mystring) - 1
mystring[last_index]
```

3.3. The middle of the string should be at `len(mystring) / 2`. However, we need to keep in mind that the *index* where a character is *must be an integer!* If you try `mystring[len(mystring) / 2]`, you'll notice that we get an error because the index we are plugging in `len(mystring) / 2` is *not* an integer![3] To fix this, we need to *cast* that *floating point number* into an *integer number,* which we can do with the `int` function.

[3] This is a `float`, which is the type used for decimal numbers. This type is automatically used for dividing integers because it may not be a whole number.

```
middle = len(mystring) / 2
mystring[int(middle)]
```